VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

# Optimalizace pomocí genetických algoritmů a genetického programování

# Optimization Using Genetic Algorithms and Genetic Programming

2014 Petr Svoboda

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

# Zadání diplomové práce

Student: **Bc. Petr Svoboda**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Optimalizace pomocí genetických algoritmů a genetického programování
Optimization Using Genetic Algorithms and Genetic Programming

Zásady pro vypracování:

Cílem práce je nastudovat a implementovat metody pro optimalizaci řešení založené na genetických algoritmech a genetickém programování (GA/GP). Výsledkem bude jak základní optimalizační metoda tak i některé její varianty nalezené v literatuře. Schopnost hledat optimální řešení bude demonstrována na klasických optimalizačních problémech a na kompresi dat, kde bude použita pro hledání optimální abecedy pro kompresi textových dat.

Práce bude obsahovat:
1. Rešerši metod založených na GA/GP a jejich variantách, včetně jejich použití.
2. Návrh implementace.
3. Otestování výsledného algoritmu na klasických optimalizačních úlohách.
4. Otestování algoritmu v oblasti komprese dat.
5. Vyhodnocení výsledků experimentů.

Seznam doporučené odborné literatury:

[1] An Introduction to Genetic Algorithms, Melanie Mitchell, MIT Press, 1998
[2] Genetic Programming: On the Programming of Computers by Means of Natural Selection, John R. Koza, MIT Press, 1993
[3] Data Compression: The Complete Reference, David Salomon, 4. ed., Springer, 2007

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Jan Platoš, Ph.D.**

Datum zadání: 16.11.2012
Datum odevzdání: 07.05.2014

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA * Fakulta elektrotechniky a informatiky * OSTRAVA *

doc. Dr. Ing. Eduard Sojka
*vedoucí katedry*

prof. RNDr. Václav Snášel, CSc.
*děkan fakulty*


Poděkování

Tato práce byla vypracována s podporou projektu Bio-inspirované metody: věda, vzdělávání a transfer znalostí, reg. č. CZ.1.07/2.3.00/20.0073 podpořeného Operačním programem Vzdělávání pro konkurenceschopnost, financovaného ze strukturálních fondů EU a státního rozpočtu ČR.

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.*

V Ostravě 7. května 2014 ..... Svoboda .....

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2014 ..... Svoboda .....

Rád bych na tomto místě poděkoval vedoucímu práce panu doc. Ing. Janu Platošovi Ph.D. za užitečné rady a čas, který mi věnoval. A své rodině za podporu při tvorbě této práce.

## Abstrakt

Tato diplomová práce je zaměřena na genetické algoritmy a genetické programování. Shrnutí jejich principů a jejich implementaci na klasickém optimalizačním problému. Dále pak experiment s jejich použitím v oblasti komprese textových souborů.

**Klíčová slova:** Optimalizace, Genetické algoritmy, Genetické programování, Komprese

## Abstract

This master thesis is focused on Genetic Algorithm and Genetic Programming. The summary of their principles and their implementation on a classic optimization problem. Furthermore, the experiment with the use of compression of text files.

**Keywords:** Optimization, Genetic Algorithm, Genetic Programming, Compression

## Seznam použitých zkratek a symbolů

| | | |
|---|---|---|
| GA | – | Genetické algoritmy |
| TSP | – | Traveling salesman problem |
| ASCII | – | American Standard Code for Information Interchange |
| PGA | – | Paralelní genetické algoritmy |

# Obsah

# Seznam tabulek

# Seznam obrázků

# Seznam výpisů zdrojového kódu

# 1 Úvod

V životě se můžeme setkat s problémy, které mají více možných řešení. Například obyčejná cesta do školy veřejnou dopravou se dá uskutečnit několika různými způsoby (výběr spojů, času odjezdu a přestupů). Ve výsledku ale záleží zejména na času příjezdu na určené místo a době strávené cestováním.

Každý si dokáže takovýto jednoduchý problém vyřešit po svém, protože možností obvykle není tak mnoho. Kdyby jich ale bylo více, nemuselo by vhodné řešení být zcela zřejmé.

Na takovéto problémy (optimalizační) se soustředí genetické algoritmy. Ty dokáží v relativně krátkém čase najít vhodné řešení i v případě, že budou tisíce či miliony možností, jak problém vyřešit.

Takový problém může představovat například dopravní dispečerský systém, který řeší pohyb lokomotiv na kolejích tak, aby byly co nejmenší provozní náklady.[14]

## 1.1 Obsah práce

Na začátku práce shrnuji základy genetických algoritmů, jejich princip, popis základního návrhu a jejich nastavení. Dále věnuji pozornost různým typům genetických algoritmů a jejich rozdílům.

V další části probírám genetické programování, jeho rozdíl oproti genetickým algoritmům a jeho základní podobu. V této části také uvádím příklad jeho užití.

Následující část obsahuje návrh genetického algoritmu a popis jeho implementace. Poté se věnuji problému komprese textových problémů, konkrétně jak využít genetické algoritmy pro optimalizaci jejich komprese.

Pak probírám klasický optimalizační problém - problém obchodního cestujícího. V této sekci popisuji jeho návrh a testuji na něm svou implementaci.

Další část obsahuje testy navrženého algoritmu pro kompresi textových souborů a jejich vyhodnocení.

# 2 Genetické algoritmy

Genetické algoritmy patří do skupiny evolučních algoritmů. Ty řeší problém pomocí metod inspirovaných biologickou evolucí.[14]

Podobně jako v přírodě, tak i v evolučních algoritmech je populace tvořena jedinci. Ti mezi sebou vzájemně bojují o přežití a reprodukci. Největší šanci přitom mají ti nejsilnější z nich, naopak přežití slabších jedinců je pouze otázkou náhody.

Genetické algoritmy se aplikují na nejrůznější problémy, které mají velký počet možných řešení (velký stavový prostor). Optimalizací se pak snaží najít to, které nejlépe splňuje zadaná kritéria (globální optimum).

## 2.1 Princip GA

V genetických algoritmech se pod pojmem jedinec (chromosome) rozumí jedno řešení zadaného problému. Toto řešení je reprezentováno geny, které určují nějaký bod stavového prostoru.

Síla, neboli kvalita jedince je určena kvalitou řešení, které pro daný problém představuje.

```
Generace g = 0
Vygeneruj populaci P(g)
Evaluace populace P(g)
Cyklus
  g = g + 1
  Vytvor novou populaci P(g)
  Vyber jedince z P(g−1)
  Aplikuj krizeni
  Aplikuj mutaci
  Evaluace populace P(g)
Dokud (ukoncovaci podminka)
```

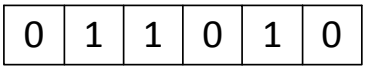
Výpis 1: Pseudokód genetických algoritmů

Jak znázorňuje pseudokód genetických algoritmů na výpisu kódu 1, na začátku se vygeneruje prvotní populace, obvykle tvořená z náhodných řešení zadaného problému. Celá populace se ohodnotí a začne se vytvářet nová generace. To znamená, že se postupně vybírají jedinci z generace předešlé a následně se na ně aplikují genetické operátory křížení a mutace. Tím vzniknou noví jedinci, kteří se začlení do nové populace. Výběr probíhá tak dlouho, dokud není nová populace zcela zaplněna, pak začne znovu ohodnocení. Proces vytváření nových generací se opakuje tak dlouho, dokud není splněna podmínka ukončení.

## 2.2 Reprezentace jedinců

Jedinci (chromosomy) jsou reprezentováni datovou strukturou, která umožňuje zakódovat potřebné informace (geny).

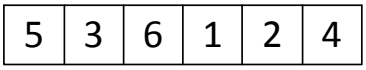
Obvykle se používá řetězec jedniček a nul, kde každý bit má nějaký význam, například barva očí.[13]




Obrázek 1: Binární řetězec

Další z používaných typů je permutace. Takový jedinec je tvořen z celých čísel jdoucích po sobě, přičemž každé číslo se může vyskytnout pouze jednou.[13]

Tento způsob se aplikuje například u problému obchodního cestujícího, kterému se věnuji v samostatné kapitole 5.




Obrázek 2: Permutace

Dále se může jednat o reprezentaci reálnými čísly nebo stromovou strukturou, a mnoho dalších.[13]

Výběr reprezentace závisí na řešeném problému.

## 2.3 Selekce

Selekce vybírá jedince pro další generace. Přednost ve výběru by měli mít jedinci s vyšší fitness hodnotou, protože mají lepší genetickou výbavu a je tedy i vyšší pravděpodobnost, že z nich křížením a mutací vzniknou lepší jedinci.[1] To ovšem neznamená, že horší jedinci by se neměli vybírat, protože i oni můžou nést podstatné genetické informace, které třeba lepším jedincům chybí.

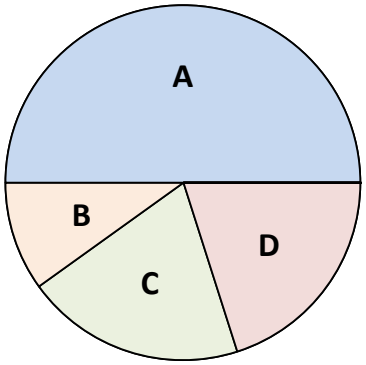
Je několik metod, které implementují tento způsob výběru. Pro svůj projekt jsem vybral následující tři.

### 2.3.1 Ruletová selekce

Jak název napovídá, tato metoda selekce je odvozena od rulety. Každý jedinec obsadí v ruletě políčka v závislosti na jeho fitness hodnotě. Pravděpodobnost výběru jedince je tedy přímo úměrná jeho kvalitě. Pomyslnou kuličkou na ruletě je vygenerování náhodného čísla, které určí, jaké políčko bude vybráno.

Tato metoda je nevýhodná, pokud se kvalita jedinců velmi liší. Jak je znázorněno na obrázku 3, jedinec A zabírá stejný počet políček jako zbylí jedinci dohromady. Častým výběrem tohoto jedince by pak mohlo dojít k předčasné konvergenci populace do lokálního suboptima.[13]

| | |
|---|---|
| Jedinec A | 5 |
| Jedinec B | 1 |
| Jedinec C | 2 |
| Jedinec D | 2 |



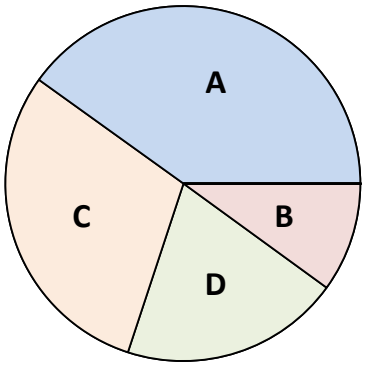

Obrázek 3: Znázornění ruletové selekce

### 2.3.2 Pořadová selekce

Pořadová selekce je podobná ruletové selekci. Políčka na pomyslné ruletě přiděluje podle pořadí jedinců v populaci seřazené podle jejich kvality. Nejhorší jedinec bude zabírat 1 políčko a nejlepší N políček. Tím se řeší problém příliš velkých rozdílů fitness hodnot, protože rozdíl mezi sousedícími jedinci bude vždy nanejvýše 1 políčko na ruletě.[13]

Obrázek 4 zobrazuje stejný příklad z předchozího obrázku 3, ale ohodnocen pomocí pořadové selekce.

| | |
|---|---|
| Jedinec B | 1 |
| Jedinec D | 2 |
| Jedinec C | 3 |
| Jedinec A | 4 |




Obrázek 4: Znázornění pořadové selekce

### 2.3.3 Turnajová selekce

Turnajová selekce na začátku vybere $n$ jedinců, kteří mezi sebou budou soupeřit o reprodukci. Turnaj vyhrává jedinec s nejvyšší fitness hodnotou.

Velikostí turnaje se dá upravit povaha turnaje. Při malém počtu výběrů mají větší šanci na reprodukci i slabší jedinci, kdežto u velkých turnajů je naopak méně pravděpodobné, že zvítězí horší jedinci.

Pokud se $n$ bude rovnat jedné, turnajová selekce se bude chovat jako náhodný výběr.[3]

### 2.3.4 Elitismus

Vlivem genetických operátorů se může stát, že nejlepší jedinci budou ztraceni při křížení nebo mutaci. Elitismus tento problém řeší výběrem $k$ nejsilnějších (elitních) jedinců, které pošle přímo do další generace. Tím se zajistí, že v každé generaci bude nejlepší jedinec v nejhorším případě stejně kvalitní, jako v generaci předchozí.[13]

## 2.4 Ohodnocovací funkce

Ohodnocovací funkce (Fitness function) odráží kvalitu jedince, tedy jak dobré řešení daného problému představuje. Pokud tato funkce bude kvalitu jedinců popisovat špatně, tak pravděpodobně ani genetické algoritmy nebudou fungovat správně, protože na ohodnocení je závislý výběr jedinců pro reprodukci.

Například u problému obchodního cestujícího je kvalita řešení určena délkou trasy.

## 2.5 Genetické operátory

Genetické operátory slouží k vytvoření nových jedinců pro další generace. Základní operátory jsou křížení a mutace.

### 2.5.1 Křížení

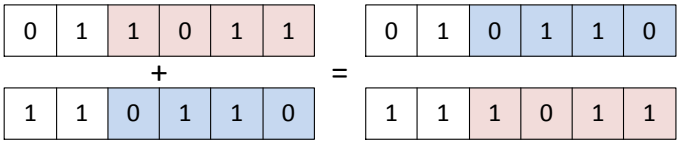
Křížení je zpravidla nejpoužívanější a nejdůležitější genetický operátor, protože v něm tkví podstata genetických algoritmů - kombinace řešení.

Křížení závisí na implementované reprezentaci jedinců, vždy ale probíhá mezi dvěma rodiči.

U klasického binárního řetězce je více typů křížení.

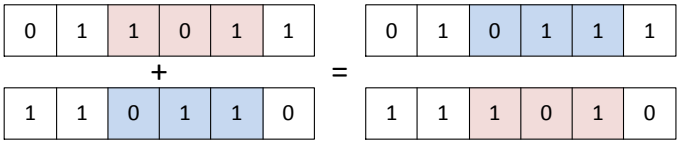
Základní a nejjednodušší je jednobodové křížení. Spočívá v náhodném výběru bodu v řetězci, vše před tímto bodem se přenese z prvního rodiče, zbytek pak z druhého rodiče. Obdobně se může vytvořit druhý potomek, který bude tvořen opačnými částmi rodičů.[13]

Jednoduché křížení znázorňuje obrázek 5




Obrázek 5: Příklad jednobodového křížení binárního řetězce

Dalším typem je dvoubodové křížení. Místo jednoho bodu se tak vygenerují hned dva body křížení. Jeden potomek je pak tvořen středem ohraničeným těmito body a kraji z druhého rodiče. Druhý potomek pak přesně naopak (obr. 6).[13]




Obrázek 6: Příklad dvoubodového křížení binárního řetězce

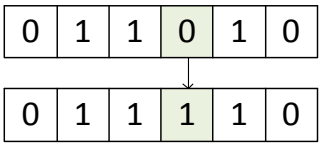
Obdobně vypadá křížení vícebodové, kde se liché části z prvního rodiče spojí se sudými částmi z rodiče druhého.[13]

Dále se pak může použít takzvané uniformní křížení, kdy má každý gen z prvního rodiče pravděpodobnost 0.5, že bude vybrán pro potomka. Pokud vybrán nebude, vezme se gen na stejné pozici z druhého rodiče.[13]

### 2.5.2 Mutace

Mutace obvykle slouží pouze pro navrácení ztracených hodnot genů do populace, aby nedošlo k předčasné konvergenci do lokálního suboptima.

U binárního řetězce mutace znamená změnu jednoho genu z nuly na jedničku a naopak, jak znázorňuje obrázek 7.[1]




Obrázek 7: Příklad mutace binárního chromosomu

## 2.6 Nastavení genetických algoritmů

Před spuštěním genetických algoritmů je třeba stanovit několik parametrů. Pro každý problém se obvykle nastavují individuálně, aby se dosáhlo co nejlepšího výkonu.

Základní parametry genetických algoritmů jsou:

- **Velikost populace**. Neměla by být příliš malá, aby jedinci svými geny pokryli celý stavový prostor. Zároveň by neměla být příliš velká, protože tím narůstá náročnost na výpočetní výkon.

  Podle výzkumu je lepší používat velké populace a menší počet generací, než menší populace a velký počet generací. [4]

- **Pravděpodobnost křížení**. Obvykle 0.75 a více.

- **Pravděpodobnost mutace**. Obvykle 0.01. Pokud se u binárního řetězce nastaví na 1.0, bude se jednat o inverzi chromosomu.

- **Metoda selekce**. Například jedna z metod popisovaných v sekci 2.3

- **Ukončovací podmínka**. Obvykle určitý počet generací, ale může být i časový limit, nebo nějaká hodnota specifického problému (např. délka trasy obchodního cestujícího).

Další parametry jsou pak například překrývání populace, nebo kolik nových jedinců se v každé generaci vygeneruje.

Pak jsou ještě parametry specifické pro každý problém. U obchodního cestujícího to je například počet měst.

## 2.7 Typy genetických algoritmů

### 2.7.1 Paralelní GA

Jak je z názvu patrné, princip tohoto typu genetických algoritmů je v rozdělení zátěže na více procesorových jednotek. Je známo několik modelů, jak tohoto dosáhnout [6]:

- **Sekvenční model** - má jedinou populaci

- **Paralelní model** - má více populací, nebo jednu populaci rozdělenou na více subpopulací

Dále se pak paralelní genetické algoritmy (PGA) dělí na různé druhy podle implementace [6]:

- **Globální paralelizace**. Genetické algoritmy mají jedinou populaci. Paralelizace spočívá v ohodnocování populace, jelikož obvykle není na ničem závislá, může se paralelně ohodnocovat více jedinců najednou. Možné je také paralelizovat genetické operátory.

- **Hrubě dělené PGA**. Populace je rozdělena na více subpopulací (kmenů). Každý kmen se vyvíjí nezávisle od ostatních. Občas dochází k výměně jedinců mezi kmeny (migrace).
- **Jemně dělené PGA**. Populace je rozdělena na hodně velice malých subpopulací. Podobně jako u hrubě dělených PGA i zde se kmeny vyvíjí nezávisle na ostatních. Tento typ PGA je navržen pro masivně paralelní architektury počítačů.
- **Hybridní PGA**. Jedná se o kombinaci všech výše zmíněných typů.

### 2.7.2 Jednoduchý GA

Populace jednoduchého genetického algoritmu se nepřekrývají. To znamená, že v každé generaci tvoří populaci zcela noví jedinci.[5]

### 2.7.3 GA se stálým stavem

Na rozdíl od předchozího případu, u genetických algoritmů se stálým stavem se generace překrývají. Noví jedinci tedy tvoří jen určitou část populace. Je třeba určit, jaké jedince mají ti noví nahradit.[5]

### 2.7.4 Inkrementální GA

Tento typ genetického algoritmu je podobný typu předchozímu. V každé generaci je ale jen velice málo nových jedinců, obvykle jeden či dva.[5]

### 2.7.5 Mikro GA

Mikro genetické algoritmy pracují s velmi malými populacemi, typicky pouze s pěti jedinci. Aby se kompenzoval nedostatek pokrytí stavového prostoru, v každé generaci se část jedinců nahradí jedinci nově vygenerovanými.[5]

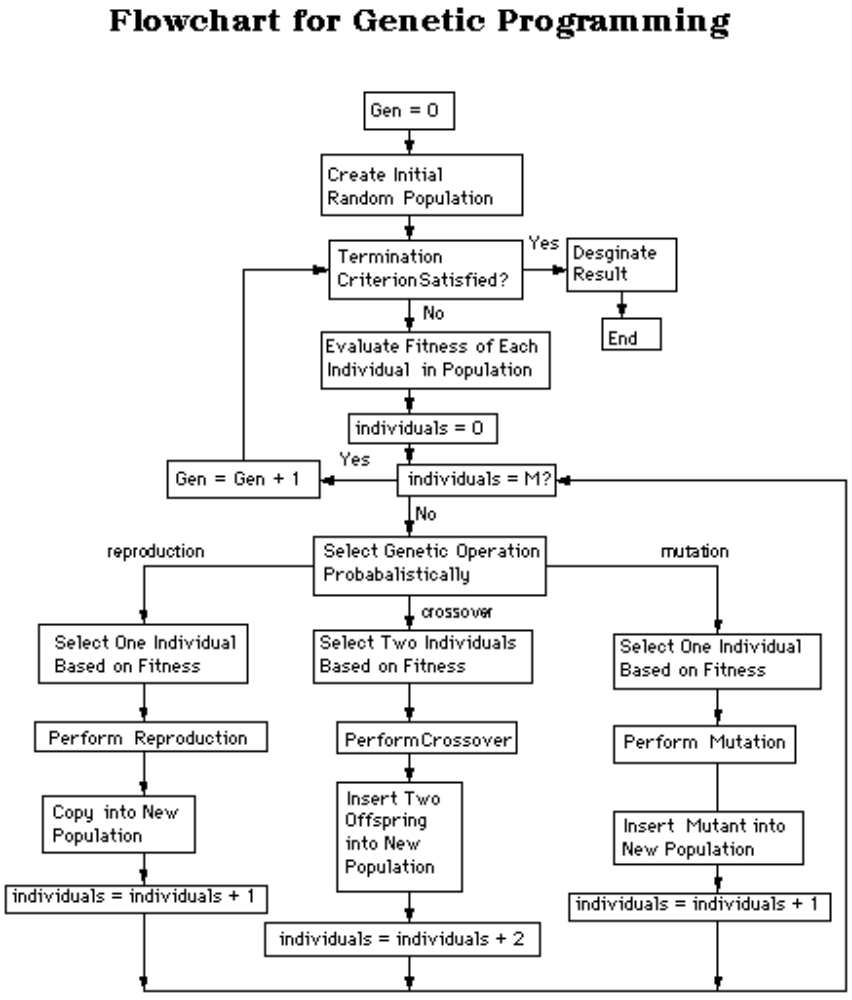
### 2.7.6 GA se stochastickým kódováním

Obvykle se používají při optimalizaci problémů s velkým počtem parametrů. Z důvodu velkého stavového prostoru by optimalizace pomocí klasických genetických algoritmů trvala velice dlouho. GA se stochastickým kódování soustřeďují prohledávání stavového prostoru do slibných míst na úkor ostatních oblastí. Prohledávání ostatních oblastí probíhá také, ale ne v takové míře.

# 3 Genetické programování

Genetické programování (GP) vyvinul John Koza v 90. letech dvacátého století. Jeho podstata je podobná genetickým algoritmům. Na začátku genetického programování je také populace tvořená jedinci, ti ale nereprezentují řešení, nýbrž počítačové programy. Jedinci se stejně jako u gentických algoritmů kříží a mutují, a vytvářejí tak nové generace. Kvalita programů je určena jejich výstupem.

Typické úlohy pro genetické programování jsou predikce, klasifikace, aproximace a tvorba programů. [8]



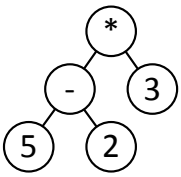

Obrázek 8: Vývojový diagram genetického programování

Jak je vidět na obrázku 8, při tvorbě nové generace může každý jedinec podstoupit jeden ze tří genetických operátorů (reprodukce, křížení nebo mutace), na rozdíl od genetických algoritmů, kde mohli jedinci podstoupit křížení a mutaci najednou.

## 3.1 Reprezentace jedinců

Reprezentace jedinců je obvykle tvořena stromovou strukturou, která se skládá z množiny terminálů a množiny funkcí.

Jako příklad je na obrázku 9 znázorněn algebraický výraz $(5-2)*3$. Množinu terminálů zde můžou představovat celá čísla, a množinu funkcí násobení, dělení, sčítání a odečítání.

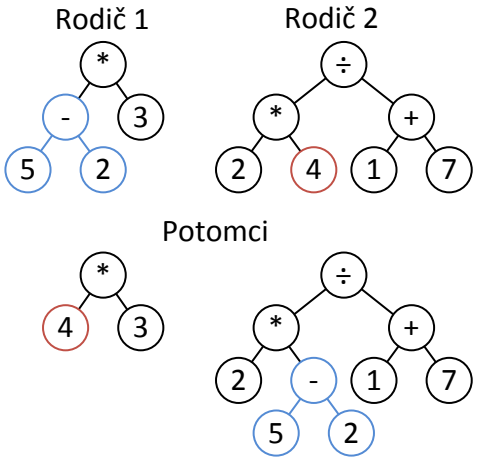



Obrázek 9: Ukázka jedince reprezentovaného stromovou strukturou

## 3.2 Křížení

Křížení stromových struktur spočívá ve výměně náhodně zvolených podstromů mezi dvěma rodiči. Výsledkem jsou pak dva potomci, kteří představují nové programy.

Nebezpečí takového křížení spočívá v rozrůstání stromů a následným zahlcením paměti počítače. Takovéto stromy se pak obvykle penalizují snížením fitness hodnoty, aby se nadále nereprodukovaly.



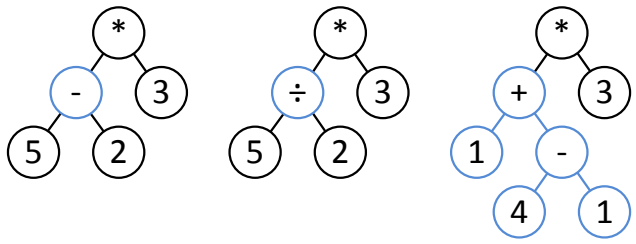

Obrázek 10: Křížení jedinců reprezentovaných stromovou strukturou

Příklad křížení dvou algebraických výrazů $(5 - 2) * 3$ a $(2 * 4) \div (1 + 7)$ znázorňuje obrázek 10. Vybrané podstromy pro křížení jsou odlišené modrou a červenou barvou. Výsledek křížení jsou dva nové algebraické výrazy $4 * 3$ a $(2 * (5 - 2)) \div (1 + 7)$.

## 3.3 Mutace

Stromové struktury mají dva různé způsoby mutace.

První způsob umožňuje zaměnit funkci za jinou funkci se stejnou aritou, a terminál za jiný terminál (obrázek 11 uprostřed). Druhý způsob je výměna náhodně vybraného podstromu za nově vygenerovaný podstrom (obrázek 11 vpravo).



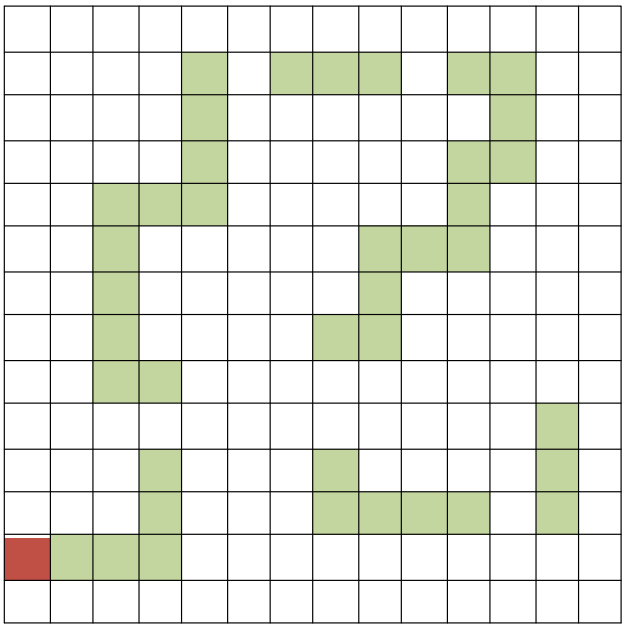

Obrázek 11: Dva způsoby mutace stromové struktury

## 3.4 Příklad genetického programování

Klasický problém pro genetické programování je stezka Santa Fe (Santa Fe Trail). Pomyslný mravenec má za úkol v daném počtu kroků (čase) sesbírat co nejvíce jídla. Mravenec přitom ví, zdali je jídlo pouze přímo před ním. Trasa jídla není spojitá, místy obsahuje mezery.[7]

Tento problém je obvykle používán pro porovnávání implementací genetického programování.

Na obrázku 12 je ukázka stezky Santa Fe. Začátek stezky je červené políčko, zelená políčka reprezentují jídlo.



Obrázek 12: Příklad stezky Santa Fe

Jak jsem popisoval v sekci 3.1, reprezentace stromovou strukturou se skládá z množiny terminálů a množiny funkcí.

Terminály představují tyto povely pro pohyb mravence [7][3]:

- **LEFT** - otočit vlevo
- **RIGHT** - otočit vpravo
- **FORWARD** - jdi vpřed

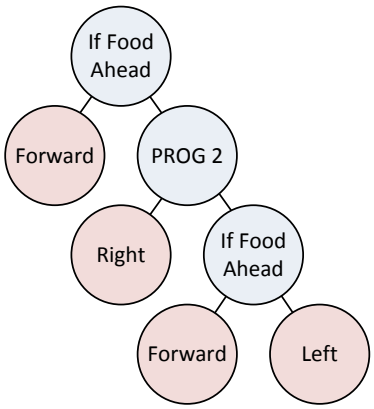
Funkce jsou pak tvořeny[7][3]:

- **If Food Ahead** - pokud je jídlo přímo před mravencem, vykonej akci prvního potomka, a pokud zde jídlo není, vykonej akci druhého potomka
- **PROG 2** - vykonej akci z prvního potomka, a poté akci z druhého v tomto pořadí
- **PROG 3** - to samé jako PROG 2, jen se nakonec vykoná ještě akce třetího potomka

Funkce PROG 2 a PROG 3 jsou do množiny funkcí přidány, aby programy nebyly příliš jednoduché, protože bez nich by měly pouze jednu funkci If Food Ahead.

Při spuštění programu se začíná v kořenu stromu. Když nezbývá žádná další akce, začíná se znovu od kořene stromu. Ukončení procházení stromu je dáno maximálním počtem kroků (terminálů), které může mravenec vykonat, nebo sesbírání všech políček s jídlem. [7]

Na obrázku 13 je znázorněn jednoduchý program mravence. Modře jsou znázorněny funkce, červeně terminály.




Obrázek 13: Program chování mravence ve stromové struktuře

Kód algoritmu z obrázku 13 by vypadal takto:

```
if (FoodAhead)
{
  Forward();
}
else
{
  Right() ;

  if (FooDAhead)
  {
    Forward();
  }
  else
  {
    Left () ;
  }
}
```

Výpis 2: Program umělého mravence

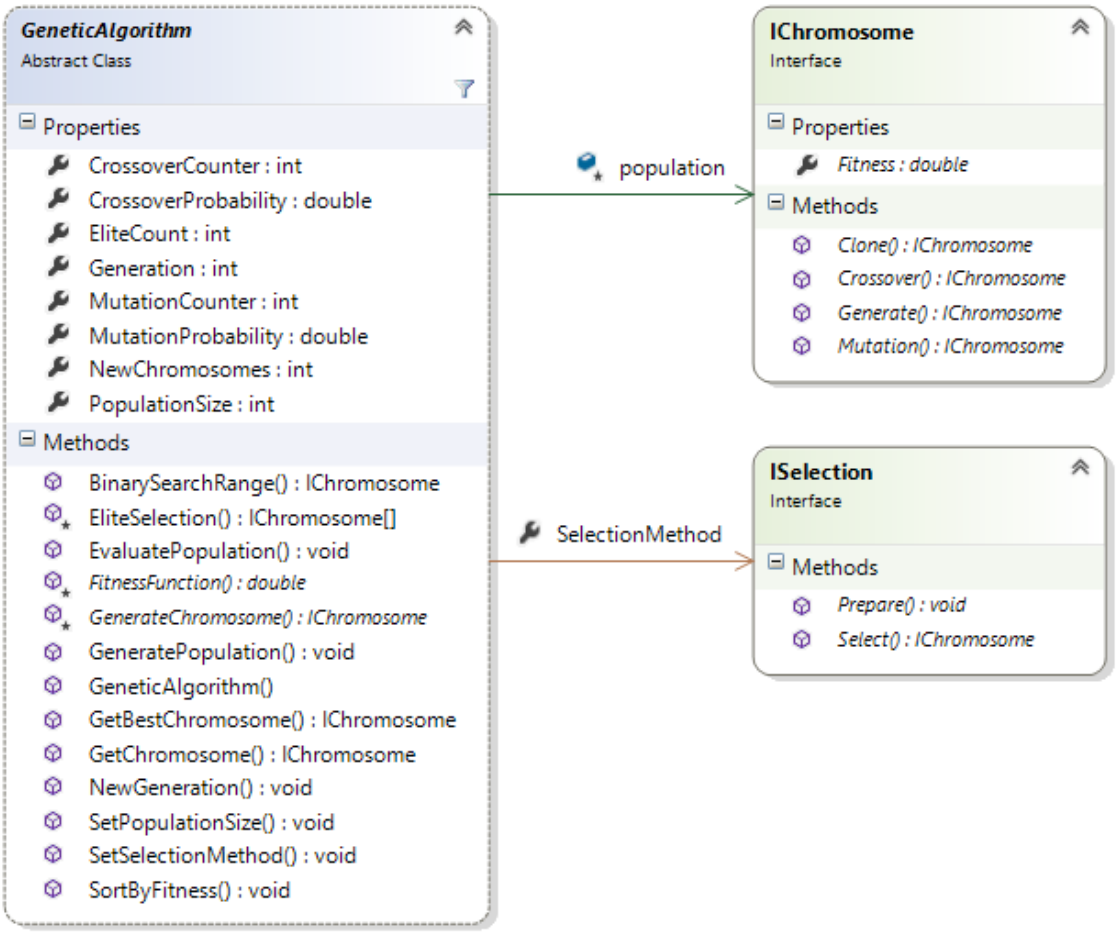
# 4 Návrh implementace

Při návrhu implementace jsem oddělil samotný genetický algoritmus tak, aby problémy které řeší byly pouze jeho rozšířením. Pro dosažení tohoto cíle jsem využil výhody objektově orientovaného programování - dědičnost a polymorfismus.

## 4.1 Genetický algoritmus

Základ genetického algoritmu jsem navrhnul jako abstraktní třídu (GeneticAlgorithm) a dvě pomocná rozhraní (IChromosome a ISelection), které se dají snadno rozšířit na řešení jakéhokoliv problému, bez nutnosti psát vše od začátku nebo zbytečného překopírovávání kódu.

Třídní diagram navržených tříd a rozhraní je zobrazen na obrázku 14.




Obrázek 14: Třídní diagram genetického algoritmu a použitých rozhraní

### 4.1.1 Abstraktní třída GeneticAlgorithm

Tato třída obsahuje základní metody algoritmu, které jsou zmíněny v jeho pseudokódu, a některá základní nastavení, která popisuji v sekci 2.6.

Třída obsahuje následující vlastnosti:

- **EliteCount** - nastavení počtu elitních prvků, které se přenesou do další generace.
- **NewChromosomes** - počet chromosomů, které se v každé generaci vygenerují
- **CrossoverProbability** - pravděpodobnost křížení
- **MutationProbability** - pravděpodobnost mutace
- **PopulationSize** - velikost populace
- **Generation** - číslo aktuální generace

a tyto metody:

- **BinarySearchRange** - rekurzivní metoda pro binární vyhledávání jedinců dle jejich fitness hodnoty. Pro nalezení není třeba znát přesnou hodnotu, což umožňuje použití vyhledávání pro Ruletovou a Ohodnocovací metodu selekce
- **EliteSelection** - protože Elitní selekce není typická selekční metoda používaná pro výběr jedinců na aplikování genetických operací, není implementována jako samostatná třída, ale jako metoda
- **EvaluatePopulation** - ohodnocení celé populace. Je zde implementována globální optimalizace, tedy ohodnocování ve více vláknech.
- **GeneratePopulation** - vygenerování (nulté) generace
- **GeneticAlgorithm** - konstruktor třídy. Má 3 volitelné parametry - velikost populace, pravděpodobnost křížení a pravděpodobnost mutace.
- **GetFittestChromosome** - výběr nejlepšího jedince z populace
- **GetChromosome** - získání ostatních jedinců
- **NewGeneration** - vytvoření nové generace (elitní selekce, vygenerování nových jedinců, selekce a genetické operátory)
- **SetPopulationSize** - pro změnu velikosti populace mimo initializaci. Pokud se populace zmenšuje, zahodí se horší jedinci. Pokud se zvětšuje, vygenerují se noví.
- **SetSelectionMethod** - nastavení metody selekce. Selekce musí být instance třídy, která implementuje rozhraní ISelection
- **SortByFitness** - seřazení populace podle fitness hodnoty

V abstraktní části jsou dvě metody:

- **FitnessFunction** - protože pro ohodnocení jedinců je potřeba znát obsah a význam chromosomu jedince, je možné ji implementovat až v třídě, která dědí z GeneticAlgorithm
- **GenerateChromosome** - vygenerování chromosomu je závislé na jeho implementaci a může být závislé na podstatě dat

### 4.1.2 Rozhraní IChromosome

Aby abstraktní třída GeneticAlgorithm mohla pracovat s jakýmkoliv typem chromosomu, navrhnul jsem toto rozhraní. Každý typ chromosomu, který bude toto rozhraní implementovat, tak bude obsahovat metody pro své naklonování, křížení s jiným chromosomem a svou mutaci.

### 4.1.3 Rozhraní ISelection

Pro snadnou implementaci nových metod selekce, jsem vytvořil rozhraní ISelection, které obsahuje definici pro dvě metody, Prepare a Select.

Metoda Prepare je určena k přípravě populace pro selekci. Například u ohodnocovací selekce (rank selection) je třeba populaci seřadit dle fitness hodnoty, a poté každému jedinci přiřadit číslo dle jeho umístění. Nakonec se provede kumulace hodnot (hodnota každého jedince je rovna součtu hodnot všech předešlých jedinců).

Metoda Select vybírá jedince z populace. Způsob výběru záleží na použitém algoritmu selekce.

### 4.1.4 Třída CryptoRandom

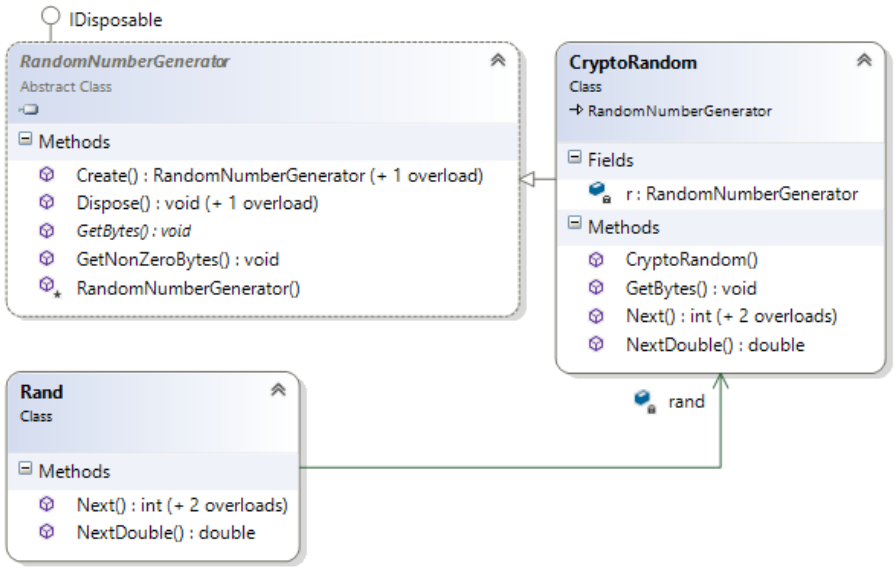
Autorem je Ben Klopfer. Třída je volně dostupná na adrese `http://thinketg.com/how-to-generate-better-random-numbers-in-c-net-2/`.

Důvodem zahrnutí do své implementace je kvalita defaultního generátoru náhodných čísel v .NET frameworku. Ten totiž používá předdefinovanou tabulku náhodných čísel, následkem čehož se může stát, že se čísla začnou opakovat. [9]

Pro lepší generování je třeba použít třídu RandomNumberGenerator ze jmenného prostoru System.SecurityCryptography. Tato třída je ale abstraktní a neumí klasické metody Next a NextDouble. Umí pouze generovat náhodná pole bytů.

Tyto nevýhody řeší třída CryptoRandom, která z RandomNumberGenerator dědí. S její pomocí poté implementuje výše zmíněné metody.

V mé implementaci jsem dále třídu CryptoRandom zaobalil do statické třídy Rand, abych mohl generovat náhodná čísla kdekoliv, bez nutnosti předávat si instanci třídy.




Obrázek 15: Třídní diagram CryptoRandom

## 4.2 Optimalizace komprese textových souborů

Komprese textových souborů musí být bezeztrátová, kdyby se nějaké informace z původního souboru ztratily, text by pravděpodobně přestal dávat smysl a komprese by tak byla nepraktická.

Způsobů jak dosáhnout bezeztrátové komprimace je více. [11]

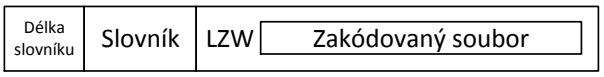
- Elementární metody (Run-Length Encoding, Braillovo písmo)
- Statistické metody (Morseův kód, Huffmanovo kódování)
- Slovníkové metody (LZ77, LZ78, LZW, LZSS)
- Ostatní metody (Data Compression using Antidictionaries)

V mém projektu používám metodu založenou na slovníku. Ty zaznamenávají do slovníku sekvence přečteného textu a jejich následný výskyt zaznamenávají pouze jako odkaz do vytvořeného slovníku.[11]

### 4.2.1 Návrh optimalizace pomocí genetických algoritmů

Genetické algoritmy pracují nad stavovým prostorem. V mém návrhu tento prostor představují kombinace všech možných dvojznaků. Nejedná se tedy o typickou metodu, kde se slovník vytváří zároveň při kompresi, ale o předdefinovaný slovník.

Po optimalizaci genetickými algoritmy jsou vybrané dvojznaky použity pro zakódování souboru. Výstup je dále zkomprimován algoritmem LZW, pro lepší výsledek komprese.




Obrázek 16: Formát výsledného komprimovaného souboru

Po optimalizaci se soubor ještě dále zkomprimuje algoritmem LZW, pro dosažení lepší komprese. Formát výsledného souboru znázorňuje obrázek 16.

### 4.2.2 Kompresní algoritmus LZW

LZW je bezeztrátový kompresní algoritmus vynalezený A. Lempelem, J. Zivem a T. Welchem v roce 1984. Je to slovníková metoda, pro kompresi tedy využívá opakovaný výskyt řetězců.[12]

Standardně pracuje s 256ti znakovou abecedou (rozšířená ASCII tabulka), kde jeden znak má 8 bitů.

Při kompresi souboru se čte vstupní soubor znak po znaku, každý výskyt nového řetězce znaků je zaznamenán do slovníku. Při dalším výskytu tohoto řetězce se místo něj na výstup zapíše jeho index ze slovníku.[12]

Slovník o omezené velikosti je na začátku komprese tvořen pouze abecedou, řetězce se do něj přidávají až za běhu. Jeho indexy potřebují pro svůj zápis více bitů, protože například pro slovník o velikosti 1024 je potřeba pro poslední index (1023) 10 bitů. Tuto nevýhodu ale kompenzuje fakt, že jeden index může představovat více znaků. [12]

Dekomprese má podobný průběh. Stejně jako komprese začíná se slovníkem tvořeným pouze vstupní abecedou. Místo znaků se na vstupu ale čtou indexy. Znaky z předchozího vstupu a první znak z aktuálního vstupu vždy tvoří nový zápis do slovníku.

Zdrojové kódy LZW algoritmu mi poskytl vedoucí práce doc. Ing. Jan Platoš, Ph.D..

**4.2.2.1 Příklad** Slovo "PEPEK"obsahuje pouze tři znaky, slovník na začátku algoritmu by tedy vypadal jako v tabulce 1.

| Index | Řetězec |
|---|---|
| 0 | E |
| 1 | K |
| 2 | P |

Tabulka 1: Počáteční slovník

Samotnou kompresi slova znázorňuje tabulka 2.

| Vstup | Aktuální řetězec | Výstup | Zápis do slovníku |
|---|---|---|---|
| P | P | - | - |
| E | PE | 2 | 3 PE |
| P | EP | 0 | 4 EP |
| E | PE | - | - |
| K | PEK | 3 | 5 PEK |
| - | K | 1 | - |

Tabulka 2: Příklad komprese LZW

Výstup LZW algoritmu by byl tedy 2031, z původních pěti znaků jsou čtyři čísla. Pokud by ale text pokračoval, a slovo ”PEPEK”by se v něm vyskytovalo znovu, už by mohlo být zapsáno pouze pomocí dvou indexů - 3 a 5.

| Vstup | Výstup | Zápis do slovníku |
|---|---|---|
| 2 | P | - |
| 0 | E | 3 PE |
| 3 | PE | 4 EP |
| 1 | K | 5 PEK |

Tabulka 3: Příklad dekomprese LZW

Zpětná dekomprese začína znovu jen se slovníkem, jehož obsah znázorňuje tabulka 1. Na konci je ale totožný se slovníkem, který se vytvořil při kompresi.

### 4.2.3 CompressionChromosome

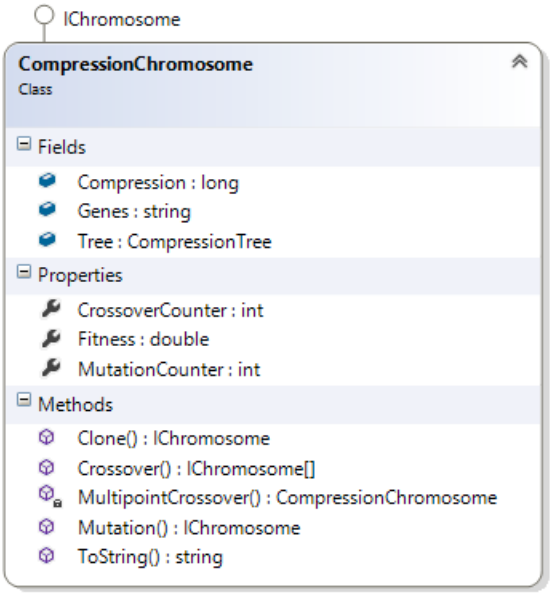
Jedná se o klasický binární řetězec. Délka chromosomu je rovna počtu bigramů ve vybraném souboru. Každý gen představuje jeden bigram. Pokud je gen nastaven na jedničku, bigram je zařazen do slovníku pro kompresi, v opačném případě je ve slovníku vynechán.

Jako mutace jedince jsem zvolil jednoduché náhodné převrácení bitu z jedničky na nulu a naopak.

Jednobodové křížení, kdy se náhodně vybere jeden bod v chromosomu, a všechny geny před tímto bodem se zkopírují z prvního rodiče a zbytek z druhého rodiče, jak znázorňuje obrázek 5, není pro tento typ problému příliš vhodné, protože chromosomy jsou často velice dlouhé. Pokud by tedy někteří jedinci měli geny potřebné k lepšímu výsledku komprese uprostřed, a naopak ty špatné geny na krajích, algoritmus by musel jejich oddělení rozdělit minimálně do dvou kroků.

Lepším řešením je vícebodové křížení. Pro křížení rozdělí chromosomy na více dílů a následně se liché pozice obsadí díly z prvního rodiče, a sudé z druhého. Jedinci si takto mohou vyměňovat libovolné geny, ať už uprostřed nebo na krajích chromosomu.

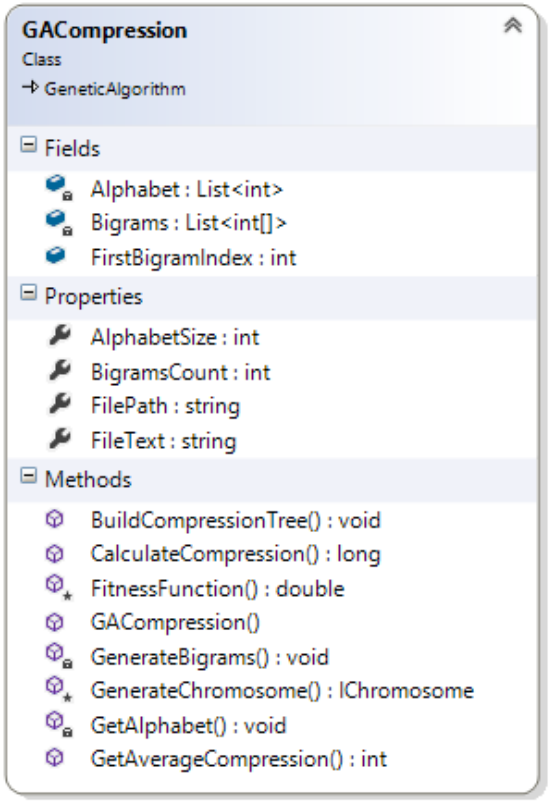
Třídní diagram CompressionChromosome je na obrázku 17.




Obrázek 17: Třída CompressionChromosome

### 4.2.4 GACompression

Tato třída je rozšíření pro abstraktní třídu GeneticAlgorithm. Kromě implementace abstraktních metod ohodnocení a generování jedinců obsahuje i metody pro vybudování kompresního stromu, výpočet komprese, načtení abecedy a dvojznaků ze souboru.




Obrázek 18: Třída GACompression

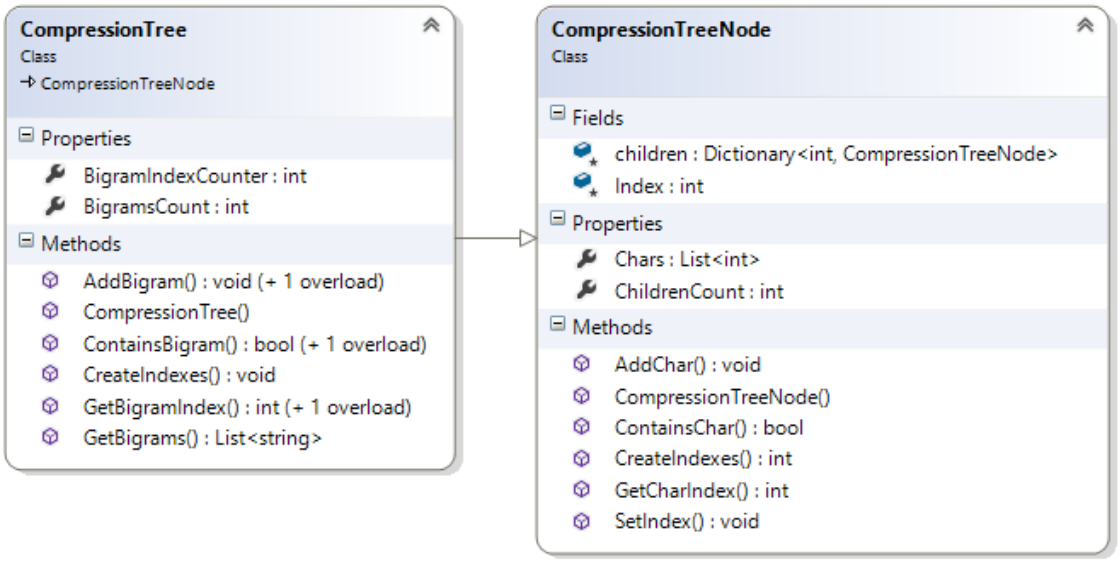
### 4.2.5 Třída CompressionTreeNode

Tato třída tvoří uzly a listy kompresního stromu. Potomci každého uzlu jsou ukládáni do slovníku (Csharp Dictionary), kde klíč tvoří znaky. Pokud je tento slovník prázdný, jedná se o list stromu.

Práci s potomky obstarávají metody AddChar - přidávání a ContainsChar - kontrola, zdali uzel má požadovaného potomka. Metoda pro odstranění potomka je zbytečná, protože při změně chromosomu se celý strom sestavuje znovu.

Každý uzel/list má proměnnou index, která značí číslo ve slovníku. Uzly v hloubce stormu 1, mají index roven hodnotě znaku v rozšířené ASCII tabulce, který představují, protože se jedná o pouhé znaky, nikoliv bigramy.

Pro práci s těmito indexy slouží metody SetIndex - nastavení indexu na hodnotu, GetCharIndex - získání indexu potomka a CreateIndexes pro očíslování svých potomků počínaje číslem zaslaným jako parametr metody.




Obrázek 19: Třídní diagram CompressionTree a CompressionTreeNode

### 4.2.6 Třída CompressionTree

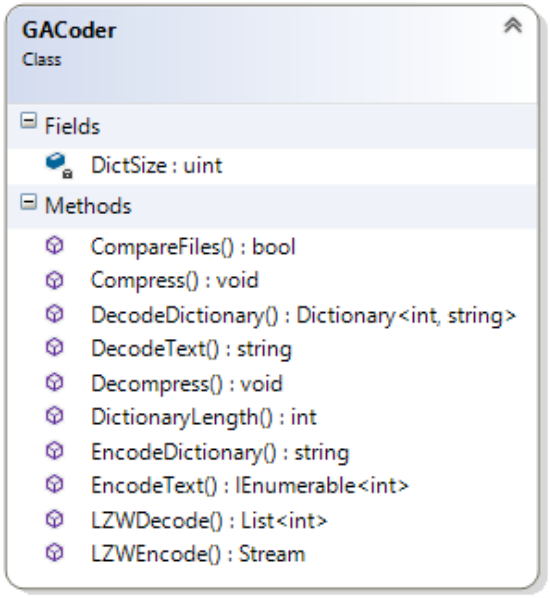
CompressionTree představuje kořen kompresního stromu. Rozšiřuje třídu CompressionTreeNode o práci s bigramy, konkrétně o jejich přidávání (AddBigram), kontrolu, zdali strom bigram obsahuje (ContainsBigram) a získání indexu bigramu (GetBigramIndex).

Pro očíslování bigramů slouží metoda CreateIndexes, která na všech potomcích třídy CompressionTree zavolá metodu CreateIndexes s parametrem, od něhož má číslování pokračovat, po očíslování je vrácen aktuální index a jeho hodnota je poslána dalšímu potomkovi. Číslování musí začínat nejméně od čísla 256, protože 0-255 jsou čísla obsazená rozšířenou ASCII tabulkou.

Kompletní výpis vlastností a metod je na obrázku 19.

### 4.2.7 Třída GACoder

Samotnou kompresi souboru zprostředkovává třída GACoder. Její metody jsou použity jak při výpočtu kvality jedince, tak při kompresi a dekompresi souboru.




Obrázek 20: Třídní diagram třídy GACoder

Komprese probíha ve dvou fázích.

První je zakódování pomocí kompresního stromu předaného z genetických algoritmů, to má na starosti metoda EncodeText. Ta jednoduše vyhledává ve stromu, jestli obsahuje dvojznak na vstupu jako bigram, pokud ano, zapíše jej jako index ze stromu, pokud ne, zapíše se pouze první znak jako hodnota v ASCII tabulce, a proces se opakuje načtením dalšího znaku.

Druhá fáze je komprese LZW algoritmem. Na začátku je důležité nastavit vstupní abecedu LZW na velikost slovníku vytvořeného optimalizací, aby algoritmus počítal i se vstupy většími než 255.

Při zápisu do souboru se na začátek zapíšou dvojznaky ze slovníku a jejich délka, a pak výstup z LZW.

### 4.2.8 Komponenta ZedGraph

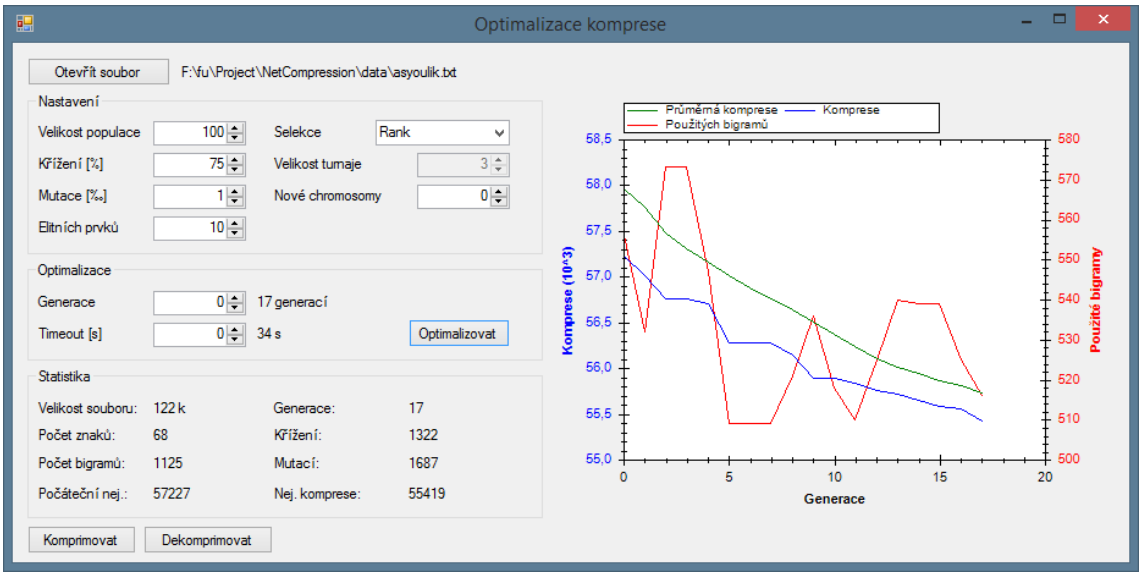
Na vykreslování grafu optimalizace jsem použil volně dostupnou komponentu ZedGraph (`http://sourceforge.net/projects/zedgraph/`). Mezi její hlavní přednosti patří jednoduchost, ale zároveň možnost přizpůsobit prakticky cokoliv.

### 4.2.9 Grafické rozhraní

Pro snadné testování optimalizace komprese pomocí genetických algoritmů jsem vytvořil jednoduché grafické rozhraní. To umožňuje vybrat soubor pro optimalizaci a měnit nastavení GA.

Jako ukončovací podmínka se dá zvolit počet generací, nebo čas, po jaký má optimalizace běžet. Dá se také nastavit ukončení pouze na manuální přerušení.

Pro znázornění optimalizace je v pravé části grafického rozhraní graf, který zachytává průměrnou kompresi, nejlepší kompresi a počet bigramů, které byly pro nejlepší kompresi použity.




Obrázek 21: Grafické rozhraní optimalizace komprese

# 5 Optimalizace problému obchodního cestujícího

Jako ukázku práce genetického algoritmu jsem vybral problém obchodního cestujícího.

Poblém obchodního cestujícího (Traveling Salesman Problem) je definován na úplném ohodnoceném grafu, kde každý jeho bod představuje jedno město a hrany představují cesty mezi jednotlivými městy. Obchodní cestující musí projít všechny města právě jednou, a vrátit se zpět na začátek trasy tak, aby délka této trasy byla co nejmenší.

Nalezení nejkratší trasy hrubou silou je pro vyšší počet měst z hlediska časové náročnosti nereálné, protože v úplném ohodnoceném grafu je celkem (N-1)! možných tras (pokud je určeno počáteční město), které prochází každý bod právě jednou. Při deseti městech, by se tak muselo zkontrolovat přes 350 tisíc tras, aby byla nalezena trasa nejkratší.

Genetické algoritmy nezaručí nalezení globálního optima, ale dokáží se mu přiblížit v daleko kratším čase.

## 5.1 Reprezentace trasy

Pro reprezentaci by se dal použít klasický binární řetězec, kde by každé město bylo reprezentováno jako $Log_2(N)$ bitů. Vznikaly by ale problémy s duplicitami. Například při mutaci jedince, by změna jednoho bitu v chromosomu mohla znamenat, že by se některé město vyskytovalo v trase dvakrát. Podobné problémy by mohly vzniknout při křížení. Musely by se tedy implementovat metody, které by takovéto nevhodné jedince opravily.[10]

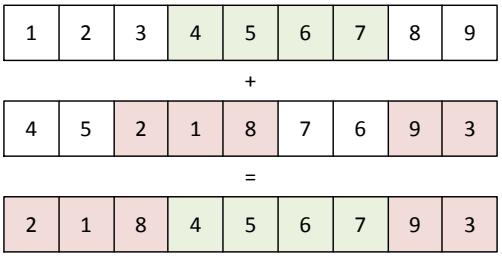
Lepší řešení představuje ukládání měst pomocí permutace (obrázek 2), kde každé číslo značí jedno město.

Jednoduchá mutace takovéto reprezentace může vypadat například tak, že se prohodí dvě náhodně vybraná města v permutaci. Tím nevznikne žádný konflikt, protože chromosome bude obsahovat ty samá čísla, pouze v jiném pořadí.[10]

Poněkud složitější je křížení. Nelze jednoduše zkopírovat část z prvního rodiče a zbytek z druhého. Některá města by se tak v chromosomu vyskytovala vícekrát, jiná naopak vůbec.

Tomuto se dá předejít použitím metody OX (Order Crossing). Ta náhodně vygeneruje dvě pozice v chromosomu, mezi kterými se vybere část z prvního rodiče, která se přenese na stejnou pozici do potomka. Doplňování zbytku pokračuje navázáním na tuto část, kopírováním měst z druhého rodiče počínaje druhým bodem křížení. Když se dojde na konec chromosomu, pokračuje se dále od začátku. Výše zmíněnou chybu opravuje kontrolou, zdali města už potomek v permutaci nemá, pokud ano, přeskočí se a pokračuje se městem dalším.[10]

Křížení metodou OX znázorňuje obrázek 22.




Obrázek 22: Příklad Order Crossing

## 5.2 Prvotní generace

Na začátku genetického algoritmu se musí vygenerovat prvotní (nultá) generace. V případě použití náhodných posloupností měst, bude velká část populace tvořena jedinci, kteří mají propojeny vzdálená města.

Lepší řešení je při generování použít heuristiku. Města se tak mohou propojit pouze s $t$ nejbližšími městy. Čím nižší parametr $t$ bude, tím lepší řešení se budou generovat. Při velmi nízkých hodnotách $t$ ale bude populace trpět nedostatkem různorodosti, a optimalizace Genetického algoritmu tak velmi brzy skončí v lokálním optimu, které může být velmi vzdálené od toho globálního.[10]

## 5.3 Ohodnocení jedinců

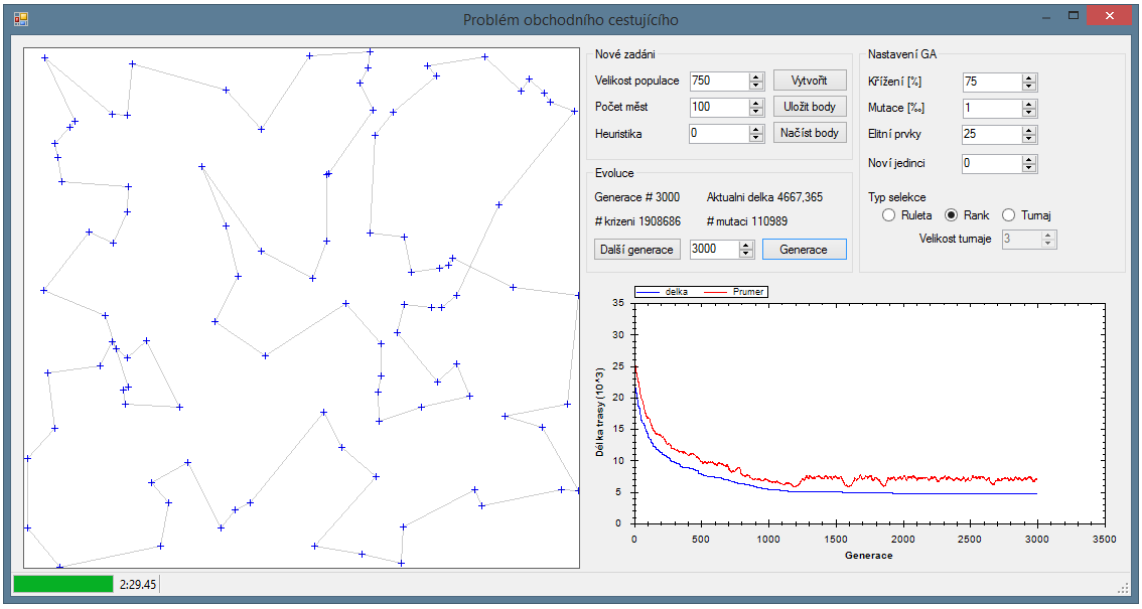
Ohodnotit populaci problému obchodního cestujícího je jednoduché. Kvalita jedince je nepřímo úměrná délce trasy, kterou reprezentuje. Pro potřeby genetického algoritmu tedy $fitness = \frac{1}{delka}$, protože fitness hodnota je obvykle přímo úměrná kvalitě.

## 5.4 Grafické rozhraní

Navržené grafické rozhraní umožňuje zvolit počet měst, heuristiku a všechny základní parametry genetického algoritmu (velikost populace, pravděpodobnost křížení a mutace, počet elitních prvků a metodu selekce a počet nových chromosomů v každé generaci).

Města jsou reprezentována body v dvourozměrném prostoru, vzdálenost mezi nimi je určena vzdáleností bodů.

Pro znázornění optimalizace zaznamenává graf nejkratší a průměrnou délku trasy v každé generaci.




Obrázek 23: Grafické rozhraní problému obchodního cestujícího

## 5.5 Test genetických algoritmů

Pro otestování genetických algoritmů na problému obchodního cestujícího jsem vybral několik scénářů. Cílem bylo najít nejlepší parametry pro optimalizaci.

Pokud u testů neuvádím jinak, tak je při testování použito nastavení, které uvádí tabulka 4.

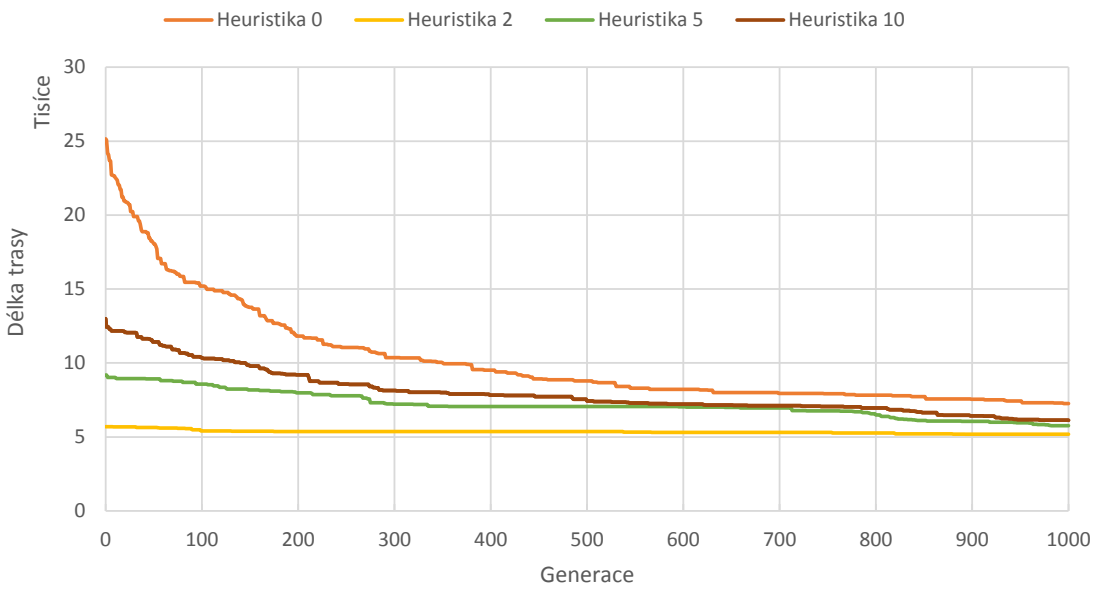
| | |
|---|---|
| Počet měst | 100 |
| Velikost populace | 250 |
| Počet generací | 1000 |
| Pravděpodobnost křížení | 0.75 |
| Pravděpodobnost mutace | 0.001 |
| Elitních prvků | 25 |
| Nové chromosomy | 0 |
| Heuristika | vypnuto |
| Metoda selekce | Rank selection |

Tabulka 4: Defaultní natsavení GA

Všechny testy se stejným počtem měst byly prováděny na stejném grafu.

### 5.5.1 Test heuristiky

Testování probíhalo na 100 městech s populací o 250ti jedincích. Pro test jsem zvolil čtyři nastavení heuristiky - 2, 5 a 10 nejbližších měst a bez heuristiky.




Obrázek 24: Graf testování heuristiky

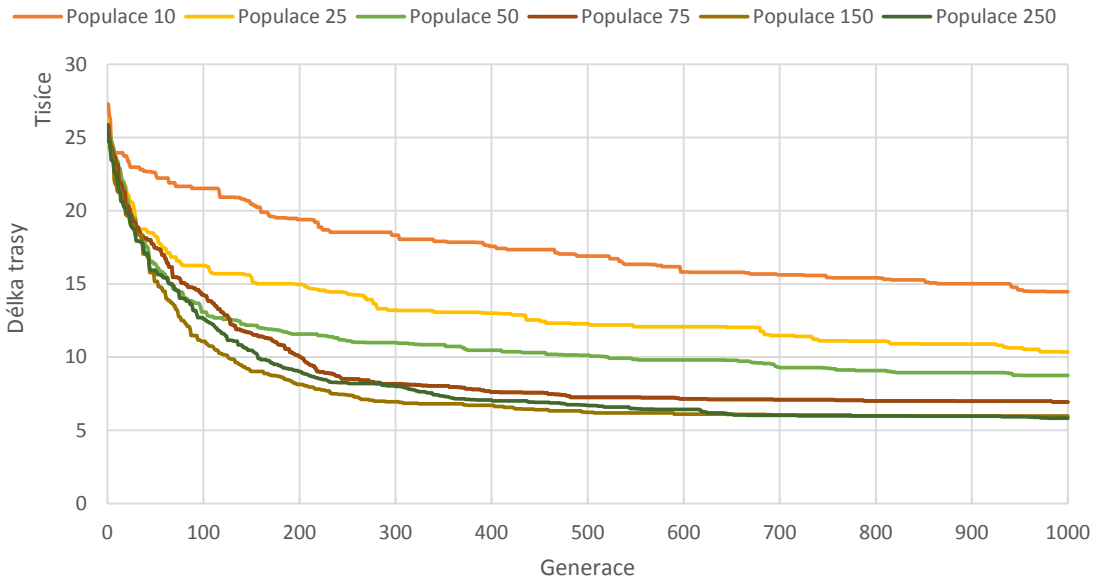
Z grafu na obrázku 24 je vidět, že čím nižší je parametr heuristiky, tím kvalitnější trasy se generují, ale o to méně efektivní je optimalizace genetických algoritmů.

Při nastavení parametru heuristiky na 10 nejbližších měst, byl v nulté generaci nejlepší jedinec skoro o polovinu kvalitnější než v té bez heuristiky, při parametru 2 dokonce o $\frac{4}{5}$. Rozdíly mezi různými nastaveními se ale po tisíci generacích snížily na zhruba sedminu až čtvrtinu optimalizace bez heuristiky.

### 5.5.2 Test vlivu velikosti populace na optimalizaci

Větší populace by logicky měly mít větší rozmanitost jedinců, a tudíž i lepší základ pro optimalizaci. S velikostí populace ale narůstá i výpočetní čas. Smysl testu je tedy vyzkoušet, jestli je použití velkých populací z hlediska optimalizace výhodné.

Testované hodnoty velikosti populace jsou 10, 25, 50, 75, 150 a 250. Počet elitních prvků byl nastaven na 10% velikosti populace.




Obrázek 25: Graf vlivu velikosti populace na kvalitu řešení

Graf měření na obrázku 25 potvrzuje, že s rostoucí velikostí populace se zlepšuje optimalizace. Nicméně to platí jen do určitého bodu, v mém testu to byla hodnota 250, při které byla výsledná optimalizace jen nepatrně lepší, s populací o velikosti 150 jedinců.

### 5.5.3 Test selekcí

V testu jsem zahrnul tři metody selekce, které jsem naimplementoval - ruletová, ohodnocovací (rank) a turnajová. Jejich popisem se zabývám v sekci 2.3.

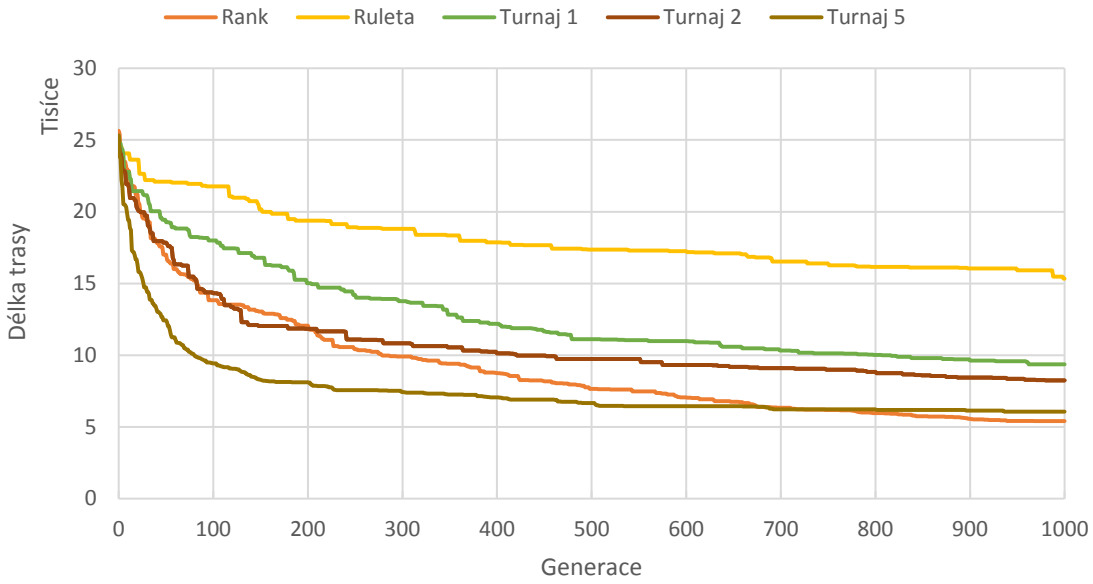
Na grafu z obrázku 26 je vidět, že ruletová selekce po tisíci generacích dosáhla zdaleka nejhorších výsledků.

Naopak turnaj o velikosti 5 dokázal za velmi malý počet generací dosáhnout kvalitní optimalizace. Nejlepšího výsledku ale dosáhla ohodnocovací selekce, která dokázala optimalizovat trasu obchodního cestujícího na $\frac{1}{5}$.

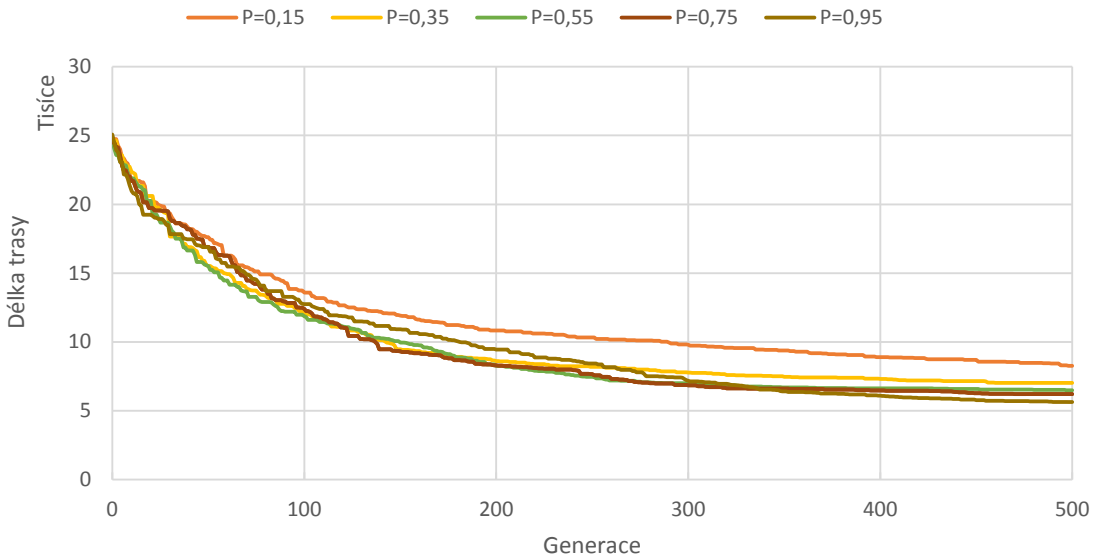
### 5.5.4 Test vlivu pravděpodobnosti křížení na optimalizaci

V tomto testu jsem se snažil nalézt nejlepší nastavení pravděpodobnosti křížení pro optimalizaci.

Z výsledků testu, který znázorňuje graf na obrázku 27 vyšlo najevo, že dopad na nalezení nejkratší délky trasy problému obchodního cestujícího je minimální. Pouze v případě nastavení pravděpodobnosti na 0,15 byl rozdíl zřetelně viditelný.




Obrázek 26: Test metod selekce




Obrázek 27: Test vlivu pravděpodobnosti křížení na optimalizaci

### 5.5.5 Vyhodnocení testů

Z testů vyšlo najevo, že nejlepší nastavení pro optimalizaci problému obchodního cestujícího o 100 městech by mělo být při použití ohodnocovací nebo turnajové selekce, s velikostí populace 150 a více. Dále použití křížení s pravděpodobností okolo 0,75.

Heuristika se dá použít na úkor optimalizace. Při jejím použití totiž populace rychleji dojde k lokálnímu suboptimu. Samotná heuristika by mohla sloužit jako taková pro řešení problému obchodního cestujícího. Její zjevnou nevýhodou je ale nutnost setřídit vzdálenosti pro každé město do všech ostatních měst.

Různé nastavení pravděpodobnosti mutace jsem do testování nezahrnul zejména z toho důvodu, že je určena pouze pro navrácení ztracených hodnot genů do populace. Při nastavení vyšší pravděpodobnosti by prohledávání stavového prostoru připomínalo spíše generátor náhodných řešení.

# 6 Test optimalizace komprese

Implementované řešení jsem testoval na několika textových souborech různého obsahu, které mi za tímto účelem poskytnul vedoucí práce pan doc. Ing. Jan Platoš, Ph.D..

Nastavení genetických algoritmů jsem měnil podle testů v sekci 5.5 a dle vlastního uvážení, abych dosáhl co nejlepší komprese s ohledem na složitost výpočtu. Všechny testy mají stejné základní nastavení, které je uvedeno v tabulce 5.

| | |
|---|---|
| Pravděpodobnost křížení | 0.75 |
| Pravděpodobnost mutace | 0.001 |
| Počet elitních prvků | 10% velikosti populace |
| Nové chromosomy | 0 |

Tabulka 5: Defaultní natsavení GA pro test komprese

V tabulkách u jednotlivých testů uvádím zbylé parametry GA a tyto naměřené výsledky:

- Komprese - nejlepší dosažená komprese
- Použitých bigramů - počet dvojznaků, které byly použity pro nejlepší kompresi
- Optimalizace - rozdíl mezi nejlepšími kompresemi v nulté a v poslední generaci
- Čas - doba trvání výpočtu (na počítači s procesorem Intel(R) Core(TM) i3-4000M CPU @ 2.40GHz)

## 6.1 Komprese souboru alice29.txt

Obsah souboru alice29.txt je anglický text Alenky v říši divů od Lewise Carrolla. Jedná se tedy o klasický text oddělovaný odstavci.

- Velikost: 152kB
- Abeceda: 74 znaků
- Počet dvojznaků: 1285

Nejlepší komprese a zároveň nejlepší optimalizace bylo dosaženo u 4. nastavení (tabulka 6), při použití větší populace. V téměř polovičním čase bylo ale 1. nastavení jen o 420 bajtů horší.

## 6.2 Komprese souboru asyoulik.txt

Obsah souboru tvoří anglický scénář komedie Jak se vám líbí od Williama Shakespeara. Často se v něm opakují jména postav a ve větší míře se v něm vyskytují bílé znaky (tabulátory a nové řádky), jinak je to vesměs anglický text.

| # | Velikost populace | Počet generací | Metoda selekce | Komprese [kB] | Použitých bigramů | Optimalizace [kB] | Čas [s] |
|---|---|---|---|---|---|---|---|
| 1 | 250 | 250 | Rank | 57,95 | 399 | 5,59 | 1933 |
| 2 | 250 | 250 | Ruleta | 58,69 | 502 | 4,57 | 1871 |
| 3 | 250 | 250 | Turnaj 5 | 58,10 | 352 | 5,42 | 1891 |
| 4 | 500 | 250 | Turnaj 5 | 57,53 | 344 | 5,90 | 3814 |
| 5 | 500 | 250 | Turnaj 10 | 58,47 | 370 | 4,92 | 3889 |

Tabulka 6: Komprese souboru alice29.txt

- Velikost: 125kB
- Abeceda: 68 znaků
- Počet dvojznaků: 1125

Rozdíl mezi nejlepší a nejhorší výslednou kompresí je necelých 600 bajtů. To je zhruba $\frac{5}{1000}$ původní velikosti souboru. Pro nejlepší efektivitu (optimalizace/čas) bych tedy zvolil druhé nastavení (tabulka 7).

| # | Velikost populace | Počet generací | Metoda selekce | Komprese [kB] | Použitých bigramů | Optimalizace [kB] | Čas [s] |
|---|---|---|---|---|---|---|---|
| 1 | 125 | 250 | Rank | 53,30 | 374 | 3,60 | 763 |
| 2 | 125 | 250 | Turnaj 3 | 53,09 | 342 | 4,07 | 762 |
| 3 | 250 | 250 | Rank | 52,72 | 328 | 3,58 | 1619 |
| 4 | 250 | 250 | Turnaj 5 | 52,96 | 317 | 4,11 | 1574 |

Tabulka 7: Komprese souboru asyoulik.txt

## 6.3 Komprese souboru random.txt

Soubor je naplněn náhodně vygenerovanými znaky. Znaky netvoří žádné srozumitelné celky, výskyt dvojznaků je tedy také náhodný.

- Velikost: 100kB
- Abeceda: 64 znaků
- Počet dvojznaků: 4096

Počet dvojznaků je nejvyšší možný nad danou abecedou ($64^2 = 4096$). Chromosomy jsou tak velice dlouhé a optimalizace trvá dlouho.

Nejlepší optimalizace dosáhlo nastavení č. 3 (tabulka 8). Výpočet trval více než dvě hodiny. Za tuto dobu se komprese vylepšila pouze o 14,63kB.

| # | Velikost populace | Počet generací | Metoda selekce | Komprese [kB] | Použitých bigramů | Optimalizace [kB] | Čas [s] |
|---|---|---|---|---|---|---|---|
| 1 | 125 | 1000 | Rank | 95,68 | 307 | 12,40 | 3835 |
| 2 | 250 | 750 | Rank | 94,17 | 155 | 13,92 | 5768 |
| 3 | 500 | 500 | Turnaj 5 | 93,37 | 86 | 14,63 | 7849 |

Tabulka 8: Komprese souboru random.txt

## 6.4 Komprese souboru alphabet.txt

Alphabet.txt je tvořen znaky anglické abecedy tak jak jdou po sobě. Celá abeceda se přitom pořád dokola.

- Velikost: 100kB
- Abeceda: 26 znaků
- Počet dvojznaků: 26

Počet dvojznaků je roven počtu znaků v abecedě, protože každý znak je v celém souboru vždy následován stejným znakem. Velikost prohledávaného stavového prostoru je tak oproti ostatním souborům velice malá.

Optimální kompresi jde přitom logicky odvodit. Pokud by měla abeceda pouze čtyři znaky a soubor by tedy vypadal takto ”abcdabcdabcd”, zcela zřejmě by nejlepší komprese byla dosažena použitím dvojznaků ”ab”a ”cd”, protože by bezezbytku pokryly celý soubor.

Pro 26 znaků je pak optimálních 13 dvojznaků. Ke stejnému výsledku dospěla i všechna testovaná nastavení v tabulce 9.

| # | Velikost populace | Počet generací | Metoda selekce | Komprese [kB] | Použitých bigramů | Optimalizace [kB] | Čas [s] |
|---|---|---|---|---|---|---|---|
| 1 | 100 | 100 | Rank | 1,99 | 13 | 0,24 | 149 |
| 2 | 100 | 60 | Ruleta | 1,99 | 13 | 0,24 | 87 |
| 3 | 100 | 30 | Turnaj 3 | 1,99 | 13 | 0,13 | 44 |

Tabulka 9: Komprese souboru alphabet.txt

## 6.5 Komprese souboru plrabn12.txt

Obsah souboru je anglická báseň Ztracený ráj od Johna Miltona.

- Velikost: 482kB
- Abeceda: 81 znaků
- Počet dvojznaků: 1192

Plrabn12.txt je největší z testovaných souborů. I při relativně malé populaci a malému počtu generací trval výpočet přes 20 minut.

Nejlepší optimalizace bylo dosaženo nastavením s největší populací a počtem generací, výpočet ale trval více jak hodinu a půl.

| # | Velikost populace | Počet generací | Metoda selekce | Komprese [kB] | Použitých bigramů | Optimalizace [kB] | Čas [s] |
|---|---|---|---|---|---|---|---|
| 1 | 125 | 125 | Rank | 200,00 | 522 | 7,87 | 1459 |
| 2 | 125 | 125 | Turnaj 5 | 199,78 | 483 | 8,46 | 1453 |
| 3 | 250 | 250 | Rank | 197,99 | 403 | 11,07 | 6065 |

Tabulka 10: Komprese souboru plrabn12.txt

## 6.6 Komprese souboru lcet10.txt

Obsahem je WORKSHOP ON ELECTRONIC TEXTS editováno Jamesem Dalym. Text je psán anglickým jazykem a členěn do odstavců.

- Velikost: 427kB
- Abeceda: 84 znaků
- Počet dvojznaků: 1934

Rozdíl mezi testováním dvou velikostí populací je vzhledem k výsledné komprimaci nepatrný (670 bajtů), rozdíl je ale znát na délce výpočtu. 670 bajtů lepší komprese znamenala o 1619 vteřin delší optimalizace.

| # | Velikost populace | Počet generací | Metoda selekce | Komprese [kB] | Použitých bigramů | Optimalizace [kB] | Čas [s] |
|---|---|---|---|---|---|---|---|
| 1 | 125 | 250 | Rank | 155,37 | 870 | 14,99 | 2604 |
| 2 | 250 | 250 | Rank | 154,70 | 809 | 14,81 | 4223 |

Tabulka 11: Komprese souboru lcet10.txt

## 6.7 Porovnání komprese

K porovnání jsem vybral nejlepší dosažené výsledky komprese na každém souboru.

Tabulka 12 obsahuje srovnání komprese algoritmu LZW s původním souborem (LZW), kompresi mého algoritmu vůči původnímu souboru (GA) a poměr komprese GA ku LZW (GA:LZW)!

Z výsledků je patrné, že nejlepší komprese se dosáhne na souboru s velmi často opakujícími se bloky textu (alphabet.txt). Naopak nejhorší komprese je na souboru s náhodným obsahem (random.txt).

V souborech s normálním textem, se kompresní poměr pohybuje okolo 40%, přičemž navrhovaný algoritmus je v průměru zhruba o 1.8% lepší než samotný algoritmus LZW.

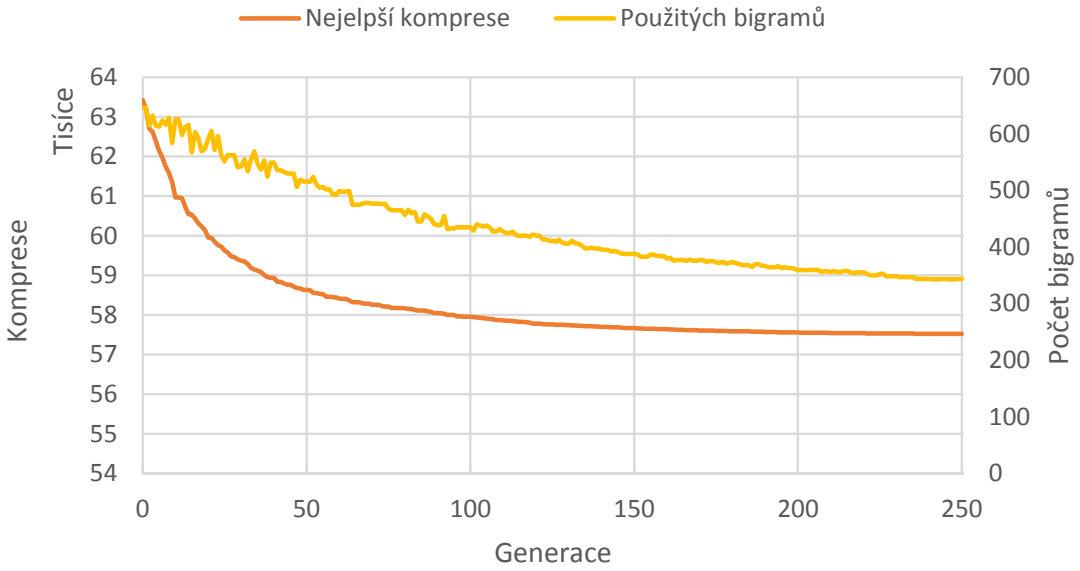
| Soubor | Velikost [kB] | LZW [kB] | LZW [%] | GA [kB] | GA [%] | GA:LZW [%] |
|---|---|---|---|---|---|---|
| alice29.txt | 152 | 62,3 | 41,0 | 57,5 | 37,8 | 92,3 |
| asyoulik.txt | 125 | 55,0 | 44,0 | 52,7 | 42,2 | 95,8 |
| random.txt | 100 | 92,5 | 92,5 | 93,4 | 93,4 | 101,0 |
| alphabet.txt | 100 | 3,1 | 3,1 | 2,0 | 2,0 | 64,5 |
| plrabn12.txt | 482 | 198,5 | 41,2 | 198,0 | 41,1 | 99,8 |
| lcet10.txt | 427 | 163,2 | 38,2 | 154,7 | 36,2 | 94,7 |

Tabulka 12: Porovnání komprese

## 6.8 Zhodnocení optimalizace

Optimalizace komprese je velmi náročná na výpočetní výkon. To je způsobeno zejména tím, že se v každé generaci pro všechny jedince vybraný soubor musí komprimovat, aby se zjistila jeho kvalita. Komprimace přitom u testů zabírá největší dobu výpočtu.

Výsledný poměr komprese přitom není v porovnání se samotným algoritmem LZW nijak vysoký.




Obrázek 28: Graf průběhu optimalizace souboru alice29.txt

Průběh optimalizace u všech souborů byl podobný. Na začátku jsou vygenerovány chromosomy se zhruba polovinou vybraných dvojznaků. V průběhu optimalizace se ale jejich počet zmenšil, protože se většina z nich překrývá, a jejich použití je z hlediska komprese nevhodné. Extrémní případ je pak u souboru random.txt, kde se mým algoritmem

dosáhlo horší komprese než u samotného algoritmu LZW. Je tak možné, že by se postupnou optimalizací došlo k výsledku, že nejlepší optimalizace se dosáhne bez použití bigramů. To by znamenalo použití pouze algoritmu LZW (+ 2 bajty pro zápis počtu použitých dvojznaků). Průběh optimalizace souboru alice29.txt znázorňuje graf na obrázku 28.

# 7 Závěr

Při tvorbě této práce jsem nabyl nových znalostí v oblasti genetických algoritmů a genetického programování, jejich princip, možnosti využití a jejich rozmanitost.

Těchto znalostí jsem využil pro implementaci základu genetických algoritmů, který jsem dále rozšířil o implementaci problému obchodního cestujícího. Na tomto rozšíření jsem úspěšně otestoval funkčnost této implementace a snažil se najít ideální nastavení genetických algoritmů.

Dále jsem základní verzi genetických algoritmů rozšířil pro použití k optimalizaci komprese textových souborů, kde jsem navrhnul algoritmus založený na slovníkových metodách. Tento algoritmus jsem otestoval na několika souborech a výsledky porovnal s algoritmem LZW. Na souborech s anglickým textem bylo dosaženo nepatrně lepších výsledků.

V budoucnu bych mohl pokračovat v rozšiřování své implementace genetických algoritmů o další metody selekce a eliminaci duplicitních jedinců. Další možnost je rozšíření na jeden z paralelních typů genetických algoritmů uváděných v sekci 2.7.1, zejména mne zaujaly Hrubě dělené PGA.

V oblasti optimalizace komprese textových souborů by se v budoucnu mohl vylepšit způsob zápisu do souboru a číslování dvojznaků pro snížení počtu potřebných bitů k zápisu indexů ze slovníku. Pro ještě efektivnější kompresi by se pak daly začlenit do komprese trojznaky (trigramy).

Petr Svoboda

# 8 Reference


[1] MITCHELL, Melanie. *Introduction to genetic algorithms.* Vyd. 1. Massachusetts: MIT Press, 1997, 209 s. First MIT Press paperback edition, 1998. ISBN 02-621-3316-4. Dostupné z: https://svn-d1.mpi-inf.mpg.de/AG1/MultiCoreLab/papers/ebook-fuzzy-mitchell-99.pdf

[2] The GP Tutorial. *The Genetic Programming Notebook* [online]. 2013 [cit. 2014-04-19]. Dostupné z: http://www.geneticprogramming.com/Tutorial/

[3] DOLAN, Kevin. *Genetic Programming Source:* Evolutionary Programming at GeneticProgramming.us [online]. 2009 [cit. 2014-04-19]. Dostupné z: http://geneticprogramming.us/

[4] TSOY, Yuri R. *THE INFLUENCE OF POPULATION SIZE AND SEARCH TIME LIMIT ON GENETIC ALGORITHM* [online]. 2003 [cit. 2014-04-19]. Dostupné z: http://qai.narod.ru/Publications/pop_size_2003.pdf

[5] POŠÍK, Petr. *Genetické algoritmy* [online]. Praha, 2000 [cit. 2014-04-21]. Dostupné z: http://labe.felk.cvut.cz/~posik/pga/theory/ga-theory.htm. Semestrální práce. České vysoké učení technické v Praze.

[6] POŠÍK, Petr. *Paralelní genetické algoritmy* [online]. Praha, 2000 [cit. 2014-04-21]. Dostupné z: http://labe.felk.cvut.cz/~posik/pga/theory/pga-theory.htm. Semestrální práce. České vysoké učení technické v Praze.

[7] *Problém umělého mravence: stezka Santa Fe.* 2000. Dostupné z: http://labe.felk.cvut.cz/vyuka/XP33ECD/GP-Mravenec.doc

[8] LEPS, Matěj. *Genetické programování* [online]. Praha, 2009 [cit. 2014-04-21]. Dostupné z: http://klobouk.fsv.cvut.cz/~leps/teaching/mmo/prednasky/prednaska12_GP.pdf. Přednáška. České vysoké učení technické v Praze.

[9] KLOPFER, Ben. *How to Generate Better Random Numbers in C# .NET.* In: Elmagine Technology Group [online]. 2014 [cit. 2014-04-21]. Dostupné z: http://thinketg.com/how-to-generate-better-random-numbers-in-c-net-2/

[10] MICHALEWICZ, Zbigniew. *Genetic Algorithms + Data Structures = Evolution Programs.* Third, Revised and Extended Edition. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996. ISBN 978-366-2033-159.

[11] KROTIL, Jiří. *Kompresní metody PPM* [online]. Praha, 2012 [cit. 2014-04-25]. Dostupné z: https://dip.felk.cvut.cz/browse/pdfcache/krotijir_2012dipl.pdf. Diplomová práce. České vysoké učení technické v Praze.

[12] BHAT, Sooraj. *LZW Data Compression.* Computer Science - Duke University [online]. 2002 [cit. 2014-04-26]. Dostupné z: https://www.cs.duke.edu/csed/curious/compression/lzw.html

[13] OBITKO, Marek. *GENETIC ALGORITHMS* [online]. 1998 [cit. 2014-04-27]. Dostupné z: http://www.obitko.com/tutorials/genetic-algorithms/index.php

[14] TEDA, Jaroslav *Genetické algoritmy a jejich aplikace v praxi.* Programujte.com [online]. 2005, č. 1 [cit. 2014-05-05]. Dostupné z: http://programujte.com/profil/164-rndr-jaroslav-teda-ph-d/